8

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/662,399	09/14/2000	Nagender P. Vedula	MS147164.1	1314

27195 7590 12/28/2004

AMIN & TUROCY, LLP
24TH FLOOR, NATIONAL CITY CENTER
1900 EAST NINTH STREET
CLEVELAND, OH 44114

EXAMINER	
BASOM, BLAINE T	
ART UNIT	**PAPER NUMBER**
2173	

DATE MAILED: 12/28/2004 12

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 10/03)

8

UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
P.O. BOX 1450
ALEXANDRIA, VA 22313-1450
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 09/662,399
Filing Date: September 14, 2000
Appellant(s): VEDULA ET AL.

MAILED
DEC 2 8 2004
Technology Center 2100

Himanshu S. Amin
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed 2/13/2004.

(1) Real Party in Interest

A statement identifying the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

A statement identifying the related appeals and interferences which will directly affect or be directly affected by or have a bearing on the decision in the pending appeal is contained in the brief.

(3) Status of Claims

The statement of the status of the claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Invention

The summary of invention contained in the brief is correct.

(6) Issues

The appellant's statement of the issues in the brief is correct.

(7) *Grouping of Claims*

The rejection of claims 1-45 stand or fall together because appellant's brief does not include a statement that this grouping of claims does not stand or fall together and reasons in support thereof. See 37 CFR 1.192(c)(7).

(8) *Claims Appealed*

The copy of the appealed claims contained in the Appendix to the brief is correct.

(9) *Prior Art of Record*

5,734,905	OPPENHEIM	3-1998
5,778,227	JORDAN	7-1998
6,496,870	FAUSTINI	12-2002

"The Component Object Model Specification, Part I: Component Object Model Introduction." Version 0.9. October 24, 1995. Microsoft Corporation. [retrieved on 2/26/2003]. Retrieved from the Internet <URL: http://activex.adsp.or.jp/Japanese?Specs/COM_Spec/COMCH01.Htm>.

(10) *Grounds of Rejection*

The following ground(s) of rejection are applicable to the appealed claims:

Claims 16, 23, 30, 32-35, 37, 38, 41-43, and 45 are rejected under 35 U.S.C. 102(b), as being anticipated by U.S. Patent No. 5,734,905 to Oppenheim. This rejection is set forth in a prior Office Action, mailed on 3/12/2003.

Claims 1-15, 17-22, 24-27, and 44 are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 5,734,905 to Oppenheim, in view of Microsoft's "Component Object Model" specification. This rejection is set forth in a prior Office Action, mailed on 3/12/2003.

Claims 28, 29, and 31 are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 5,734,905 to Oppenheim, in view of U.S. Patent No. 5,778,227 to Jordan. This rejection is set forth in a prior Office Action, mailed on 3/12/2003.

Claims 36, 39, and 40 are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 5,734,905 to Oppenheim, in view of U.S. Patent No. 6,496,870 to Faustini. This rejection is set forth in a prior Office Action, mailed on 3/12/2003.

(11) Response to Argument

Regarding claims 16, 23, 30, 32-35, 37, 38, 41-43, and 45, the appellant argues that Oppenheim (U.S. Patent No. 5,734,905) does not teach or suggest each and every element of the claimed invention, and that particularly, Oppenheim does not disclose a function object utilized to...create a mapping...between a source object and a target object...by associating a source object node with a target object node via the function object, as is expressed by each of these claims. The Examiner respectfully disagrees with this argument. As asserted by the appellant, Oppenheim discloses that two or more unrelated application programs may be linked in order to exchange data. Oppenheim particularly discloses that this is done by graphically connecting "application program objects," each understood to represent an instance of an application:

> *FIG. 8 shows an example of performing object transformations for the purpose of linking two or more objects 270, 272, 274 so that those objects will work together as though they were a single object. The linked objects are typically application program objects that may or may not have been specifically written to work with one another. This type of "linking object transformation" is typically invoked whenever a first program object 270 is used to transform a second program object 272, and the second program object 272 has at least one defined input/output port 274 to which the first program object 270 can be linked.*
>
> *The purpose of the object linking process is to cause data flowing through one object to be automatically routed to another object for further processing.*
>
> *In the example shown in FIG. 8, the first program object 270 is an analog-to-digital data converter (A/D Converter) object 270, and the second program object 272 is a signal processor object 272. The signal processor object 272 is shown as having four input/output ports 274-1 through 274-4. To initiate the object transformation the user "slaps" the A/D converter object 270 onto the signal processor object 272. If object 272 has only one input signal port, then the A/D converter object 270 would be automatically linked by the transform execution program of the A/D converter object 270 to that port.*

> *Otherwise, the user will be requested by the A/D converter object's transform execution program 175 (see FIG. 2) to select the input port of the signal processor object 272 to which the A/D converter object should be linked. In the example in FIG. 8, the result of the object linking process is the linking of the A/D converter object 270 to port 274-2 of the signal processor object 272.*
>
> *A second object transformation shown in FIG. 8 is initiated by "slapping" filter object 276 onto signal processor object 272, causing filter object 276 to be linked to output port 274-3 of the signal processor object. Links between objects are stored as "program links 176" (see FIG. 2) as part of the transform execution interface of the corresponding application programs. The resulting linked objects 270, 272 and 276 will operate together so that data flowing into the A/D converter object 270 will be directed to the signal processor 272 after conversion by object 270, and so that data output by the signal processor 272 is directed to the input of filter object 276.* (See column 8, lines 22-65).

Thus in the example of figure 8, data processed by the "A/D Converter" object is routed to the "Signal Processor" object, whereby it is processed via some sort of signal processing function, and then routed to a "Filter" object for further processing:

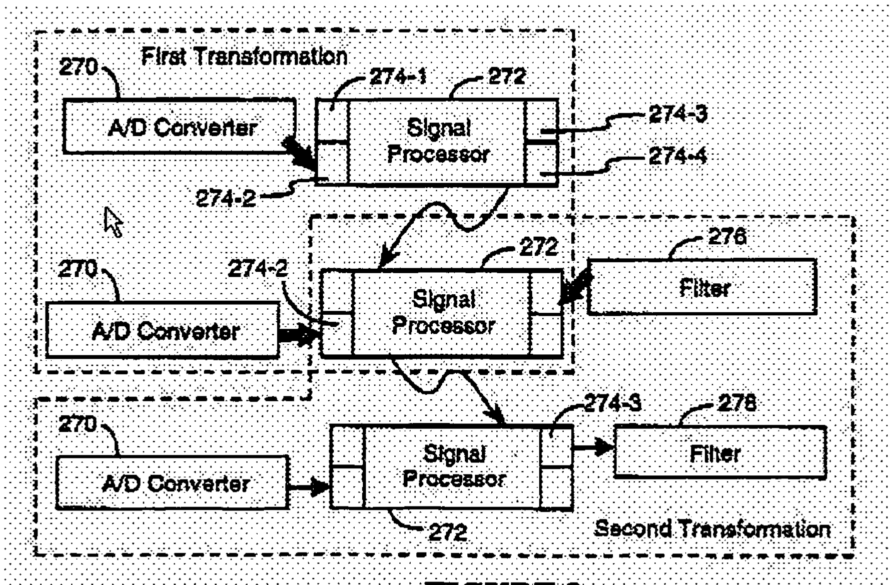

FIGURE 8

As asserted in each of the previous Office Actions, dated 3/12/2003 and 9/10/2003, the Examiner considers this Signal Processor object a "function object," like that expressed in at least independent claims 16, 32, 33, 41, 42, 43, and 45, as the Signal Processor object is implemented to create a mapping to direct data between a source object, specifically an A/D converter, and a target object, specifically a Filter. The appellant, however, alleges that equating this Signal Processor object to the claimed function object is inconsistent with the definition of "function object" given in the specification, that being an "elemental unit of functional transformation." Again, the Examiner respectfully disagrees with this argument, and asserts that the Signal Processor object of Oppenheim is in fact "an elemental until of functional transformation," given the broadest, most reasonable interpretation of such "an elemental unit of functional transformation." The Signal Processor object is considered an elemental until, as it is a basic constituent, which may be linked with other objects to act as a single object:

> *FIG. 8 shows an example of performing object transformations for the purpose of linking two or more objects 270, 272, 274 so that those objects will work together as though they were a single object. The linked objects are typically application program objects that may or may not have been specifically written to work with one another.*

Additionally, the Signal Processor object is considered a unit of functional transformation, as it performs a function, specifically a signal processing function, on data at its input (received from the A/D Converter). The Signal Processor thus transforms the data, making the transformed data available at its output (for reception by the Filter). Consequently, the Signal Processor object of Oppenheim is considered an elemental unit of functional transformation, or in other words, a function object. The Examiner further notes that the definition of a function object given in the specification, that being an elemental unit of functional transformation, is a relatively broad definition. For example, the A/D Converter and Filter objects of Oppenheim may each also be considered a function object, as they are each constituent objects which act to perform a function on data, thus transforming the data. Therefore, and even with the incorporation of material from the specification, Oppenheim teaches function objects and the implementation of such function objects.

Further regarding the Signal Processor of Oppenheim, the appellant submits that, because this Signal Processor is an application program which can act alone, it cannot be a function object:

> *As the **application programs** disclosed in Oppenheim are designed for a particular task and can act standing alone, they cannot be "an elemental unit of functional transformation", the definition of the claimed **function object** as defined in the specification.* (See page 5 of the appellants appeal brief).

The Examiner respectfully disagrees with this argument, understanding that the ability to act standing alone does not necessarily preclude an application program from being an elemental unit of functional transformation. It is understood that such features are not mutually exclusive, and in fact, may coexist. As described above, the Signal Processor of Oppenheim satisfies the definition of an elemental unit of functional transformation, i.e. a function object. Assuming however, that the appellant's assertion is correct – that an application program which can act standing alone cannot be an elemental unit of functional transformation (an assertion with which the Examiner does not necessarily agree), the Examiner notes that the Signal Processor of Oppenheim is not an application program, but instead an "application program object:"

> *FIG. 8 shows an example of performing object transformations for the purpose of linking two or more objects 270, 272, 274 so that those objects will work together as though they were a single object. The linked objects are typically application program objects that may or may not have been specifically written to work with one another. This type of "linking object transformation" is typically invoked whenever a first program object 270 is used to transform a second program object 272, and the second program object 272 has at least one defined input/output port 274 to which the first program object 270 can be linked.*

> *The purpose of the object linking process is to cause data flowing through one object to be automatically routed to another object for further processing.*
>
> *In the example shown in FIG. 8, the first program object 270 is an analog-to-digital data converter (A/D Converter) object 270, and the second program object 272 is a signal processor object 272.* (See column 8, lines 23-40 of Oppenheim).

Oppenheim discloses that such application program objects are distinct from application programs, and does not explicitly disclose that they may act standing alone:

> *Referring to FIG. 2, each object 160 used in the computer system is a data structure. As is standard in many object oriented computer systems, each* object includes a header 162, which may be accessible only to the operating system, and a body 163, which contains user definable data and software. The object header 162 includes a pointer 164 to the application program 165 that created the object 160.
>
> *The present invention adds a component, herein called the object transform interface or communication interface 166, to each application program 165. The object transform interface 166 can either be part of the application program of a separate software module linked to the application program. Each object can send and receive messages via the object transform interface 166 associated with the application program referenced in its header. Each object also preferably includes an object type parameter or set of parameters 168 that identify the object's type for purposes of performing object transformations.*
>
> *In the preferred embodiment of the present invention every user accessible object has an associated object transform interface 166. In alternate embodiments it is possible that only those objects which will participate in the object transformation processes of the present invention will have an associated object transform interface 166, and thus some objects in such systems will not have an object transform interface.* (See column 5, lines 12-38).

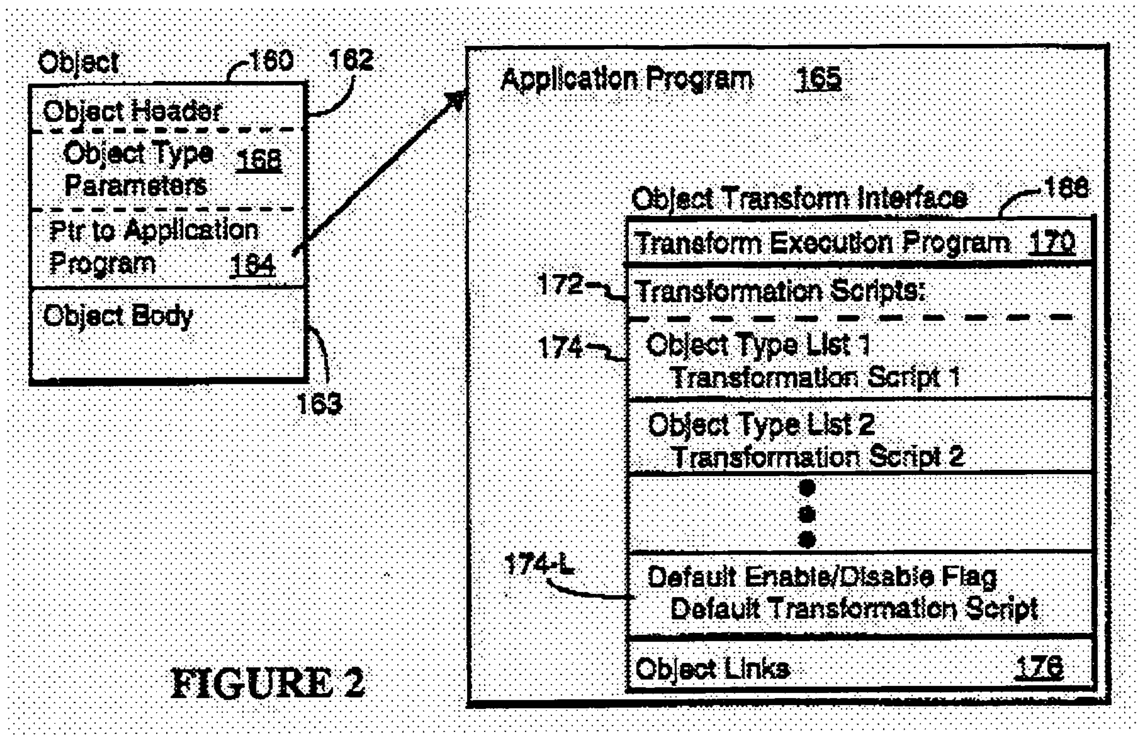
Object
160
162
Object Header
Object Type Parameters 168
Ptr to Application Program 164
Object Body
163
Application Program 165
Object Transform Interface
166
Transform Execution Program 170
172
Transformation Scripts:
174
Object Type List 1 Transformation Script 1
Object Type List 2 Transformation Script 2
174-L
Default Enable/Disable Flag Default Transformation Script
Object Links 176

FIGURE 2

Consequently, even if it is true that an application program which can act standing alone cannot be an elemental unit of functional transformation, the Signal Processor of Oppenheim is not an application program, but instead an application program object. Therefore the appellant's assertion, that an application program which can act standing alone cannot be an elemental unit of functional transformation, does not apply to the Signal Processor of Oppenheim, which is an application program object, distinct from an application program. The Examiner consequently maintains that the Signal Process of Oppenheim is an elemental unit of functional transformation, i.e. a function object.

Specifically referring to claims 16, 32, 42, and 44, the appellant again asserts that Oppenheim discloses linking three different application programs to allow data to flow from a first application program to a second application program, and from the second application program to a third application program. The appellant subsequently argues, however, that such application programs do not include nodes, and thus concludes that Oppenheim does not teach associating a source object node with an input of a function object and associating a target object node with the output of the function object, as is claimed. The Examiner respectfully disagrees with this argument. The appellant asserts that the nodes recited in the claims are structural components of particular schemas. However, the Examiner notes that such a concept is not expressed in the claims, only that a source object comprises a node, and that a target object comprises another node. Given the broadest, most reasonable interpretation of such nodes, the Examiner submits that Oppenheim presents a source object node and a target object node, and teaches associating this source object node with an input of a function object and associating the target object node with the output of the function object. As is described above, Oppenheim teaches linking an A/D Converter object to the input of a Signal Processor object, considered a function object, and linking the output of the Signal Processor object to a Filter object such that data flows from the A/D Converter to the Signal Processor and from the Signal Processor to the Filter object. The input of the Signal Processor object is consequently considered to be associated with a junction of the A/D Converter, and the output of the Signal Processor is considered to be associated with a junction of the Filter, such that data flows from the A/D converter to the input of the Signal Processor and from the output of the Signal Processor to the Filter. In other words, the input of the Signal Processor object is considered to be associated with a node of the A/D Converter and the output of the Signal Processor is considered to be associated with a node of the Filter. Assuming, however, that the appellant's assertion is correct – that the nodes, as claimed, are in fact structural components of particular schemas (an assertion with which the Examiner does not necessarily agree), the Examiner submits that Oppenheim still teaches such nodes. For example, Oppenheim discloses that objects may be comprised of multiple input and output "ports:"

> *In the example shown in FIG. 8, the first program object 270 is an analog-to-digital data converter (A/D Converter) object 270, and the second program object 272 is a signal processor object 272. The signal processor object 272 is shown as having four input/output ports 274-1 through 274-4. To initiate the object transformation the user "slaps" the A/D converter object 270 onto the signal processor object 272. If object 272 has only one input signal port, then the A/D converter object 270 would be automatically linked by the transform execution program of the A/D converter object 270 to that port. Otherwise, the user will be requested by the A/D converter object's transform execution program 175 (see FIG. 2) to select the input port of the signal processor object 272 to which the A/D converter object should be linked. In the example in FIG. 8, the result of the object linking process is the linking of the A/D converter object 270 to port 274-2 of the signal processor object 272.* (See column 8, lines 37-53).

Additionally, it is understood that such ports may be associated with a particular variable of a schema employed in an object:

> *For instance, a user might want to connect an input port of his/her computer to an analog-to-digital data conversion program, and then want to connect the output from the conversion program to a particular variable or port of another program. Making such connections between different application programs, especially if the application programs were not specifically written to be used together, is generally something that requires considerable programming expertise when using current computers.*
>
> *It is a goal of the present invention to make it possible for non-programmers to cause separate application programs to interact in ways that are intuitive and useful.* (See column 3, lines 15-30 of Oppenheim).

Such ports are consequently considered structural components of a particular schema, and it is understood that these ports may be added to objects, like the A/D Converter and Filter object described above.

With particular respect to claims 33, 41, 43, and 45, the appellant submits that Oppenheim teaches creating an object containing an appropriate mathematical function, but asserts that such teachings are only in connection with "transformer objects," which do not have an input and an output. The appellant asserts that the application program objects described above, which do comprise an input and an output, are pre-existing and consequently not able to be created. Accordingly, the appellant argues that Oppenheim does not disclose creating a script component having computer-executable instructions for performing a function using the user interface in connection with creating a graphical component associated with the function and having an input and an output, as is claimed. The Examiner respectfully disagrees with this argument. As submitted by the appellant, Oppenheim teaches creating a script component having computer executable instructions for performing a function using the user interface:

> *In the example shown in FIG. 5, the transformer object 230 is a data filter, which would typically be implemented by creating an object containing an appropriate mathematic function using an application program such as Mathematica. The transformee object 232 contains data that can be filtered. For instance, the transformee object 232 might contain a musical score and the transformer object 230 might contain a filter function that can be used to control the amplitude of different frequency components of the musical composition represented by musical score in transformee object 232. Alternately, the transformer object 230 may contain a computer program, created using a program such as Small Talk, that can be used to modify the musical score in object 232. For instance, a simple program containing one line of code:*
>
> *event velocity: event pitch*

> *would cause the loudness (called velocity in the MIDI language used to control music synthesizers) of each musical note in the score to be modified so as to be proportional to its pitch.* (See column 7, lines 2-20).

Contrary to the appellant's assertion, it is understood that such teachings may be applied to not only create transformer objects, but also to create the above-described application program objects as well. Oppenheim teaches that the transformer objects and the above-described application program objects are essentially the same, as evidenced by the fact that Oppenheim refers to both transformer objects and the above-described application program objects in conjunction with performing "object transformations" (See column 7, lines 1-32; and column 8, lines 23-33, for example). Additionally, Oppenheim discloses that both types of objects consist of the same data structure, specifically that described in column 5, line 10 – column 6, line 2. It is consequently, understood that, since transformer objects and the above-described application program objects are essentially the same, the teachings regarding transformer objects are equally applicable to the above-described application program objects. In other words, it is understood that a user employing the teachings of Oppenheim may create a script component having computer-executable instructions for performing a function using the user interface, in conjunction with creating an application program object, i.e. function object, which comprises an input and an output. As a user is able to create such application program objects, which comprise a script component and an interface component adapted to provide the script component to a compiler or interpreter and provide the graphical component to the graphical user interface (see column 5, line 10 – column 6, line 2), Oppenheim is considered to teach associating the script component, the graphical component, and the interface component.

Regarding claims 1-15, 17-22, 24-27, 44, 28, 29, 31, 36, 39, and 40, the appellant provides similar arguments to those discussed above. The Examiner consequently maintains that the rejections of these claims are appropriate, particularly for the reasons described above.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

[illegible]

Blaine Basom
Assistant Patent Examiner
December 23, 2004

Conferees:

[illegible]

JOSEPH FEILD
SUPERVISORY PATENT EXAMINER

[illegible]

JOHN CABECA
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

AMIN & TUROCY, LLP
24TH FLOOR, NATIONAL CITY CENTER
1900 EAST NINTH STREET
CLEVELAND, OH 44114